# Laboratorio 3: "Microcontrolador MSP430"

**Objetivos.**

- Dominar el uso de un simulador assembler y herramientas para la depuración de programas.
- Utilizar las ventajas de una simulación para realizar la inspección de variables y resultados. Efectuar una traza de variables.
- Realizar la depuración de programas escritos en C. Compilar y ejecutar programas assembler paso a paso y con puntos de quiebre (breakpoints) visualizando los cambios en la ventana de registros y en las zonas de memoria.

**Preparación previa.**

## 3.1 Interfaz C – Assembler MSP430

Para el laboratorio se recomienda la lectura del apunte de Interfaz C – Assembler y MSP430 Quickstart (link: "Kit de Laboratorio") disponible en el sitio del curso, en la sección Laboratorios / Módulo MSP430.

Para el programa 3.1 responda las siguientes preguntas:

a) ¿Cuál es la dirección de inicio de la RAM?
b) ¿Cuál es la dirección de inicio de la memoria flash?
c) ¿A partir de qué dirección de memoria se almacena el código del programa?
d) ¿Qué es lo que hace la función (assembler) *memcpy*?
e) ¿Hacia donde crece el stack?
f) Cuando se invoca una subrutina con la instrucción assembler *call* ¿dónde se almacena la dirección de retorno?
g) ¿Dónde se almacena el valor de retorno del programa (instrucción *return*)?
h) ¿Cuáles son los modos de direccionamiento del MSP430?
i) ¿A qué tipo de direccionamiento corresponden los símbolos #, &, y @?

**Programa 3.1**

```
#include "msp430x14x.h"

int i1 = 1, i2 = 2, i3 = 3, i4 = 4, i5 = 5, i6 = 6;
int brr[] = {7,8,9,10,11,12};

int main(void)
{
  int arr[6];
  int i, r;

  i1 = i2 + i3;
  i2 = i1 - i2;
  i3 = i1 & i4;
  i4 = i3 | i2;
```

```
  i5 = i3 ^ i4;
  i6++;

  for (i = 0; i < 6; i++)
  {
    if (brr[i] & 2)
      arr[i] = brr[i];
    else
      arr[i] = -1;
  }

  r = -2;
  return(r);
}
```

Modifique el programa anterior, incluyendo la variable `r` como una variable global no inicializada:

j) ¿Dónde se almacena la variable r?
k) ¿Qué cambios ocurren en el código assembler?

Utilizando el compilador **IAR** cargue el programa 3.2 y responda las siguientes preguntas:

a) ¿Con qué valor se inicia el puntero pi?
b) ¿Cómo maneja el stack para almacenar los argumentos la función f1?
c) Dibuje el stack después de la invocación de la función f1.
d) Indique en qué registros se almacenan las variables locales de main.

**Programa 3.2**

```
int *pi;
int x = 1, y = 2;

int f1(int a0, int a1, int a2, int a3, int a4, int *a5)
{
  a0 = a1 + a2 + a3 + a4;
  a1 = a0 + a2;
  *a5 = a1 + a0 + (*a5);
  return(a0);
}

int main(void)
{
  register int v8 = 8, v7 = 7, v6 = 6, v5 = 5;
  register int v4 = 4, v3 = 3, v2 = 2, v1 = 1;

  pi = &x;
  y = *pi;
  v3 = f1(v8,v7,v6,v5,v4,pi);

  return(0);
}
```

**En el Laboratorio.**

Se recomienda revisar el material de apoyo, manual de usuario y hoja de datos del MSP430, disponible en el sitio.

**3.1** Dar respuesta a todas las preguntas de la preparación previa.

**3.2** Simular el programa 3.3 visualizando la ejecución del código assembler. Muestre los valores que va tomando el puerto 1.

**Programa 3.3**

```
#include  <msp430x14x.h>

void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;  // Detiene el timer watchdog
  P1DIR |= 0x01;             // Setea el pin 0 de la    puerta P1 en
                             // dirección de salida

  for (;;)
  {
    unsigned int i;

    P1OUT ^= 0x01;           // Hace un toggle del bit P1.0

    i = 50;                  // Se realiza un delay por software
    do (i--);
    while (i != 0);
  }
}
```

**3.3** Modifique el programa 3.3 de tal manera de que la salida sea por el puerto 3. ¿Qué es el watchdog timer?

**3.4** Emule el programa 3.3 utilizando el módulo del microcontrolador MSP430 del laboratorio y visualice las señales del puerto 3.

a) Emule el programa 3.4 y muestre en el osciloscopio qué ciclo de trabajo tiene la señal dada. Modifique el programa y obtenga una señal con un ciclo de trabajo de un 33%.

**Programa 3.4**

```
#include <msp430x14x.h>

void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;
  P1DIR |= 0x01;              // P1.0 salida
  CCTL0 = CCIE;               // CCR0 habilita interrupción del timer
  CCR0 = 500;
  TACTL = TASSEL_2 + MC_1;    // reloj de subsistema, modo up to CCR0

  _BIS_SR(LPM0_bits + GIE);  // modo bajo consumo

  // interrupciones
}

// Timer A0 rutina de servicio de interrupción
#pragma vector = TIMERA0_VECTOR
__interrupt void Timer_A (void)
{
  P1OUT ^= 0x0F;              // Toggle P1.0
}
```

b) Diseñe un programa que permita visualizar por el puerto 1 las señales de la figura 3.1 para una frecuencia de 285,72 [Hz].

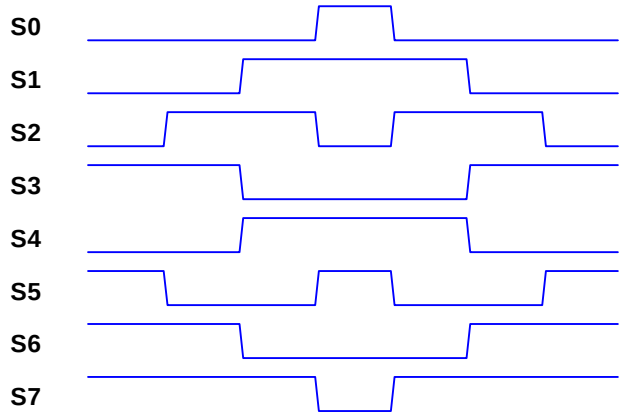


**Figura 3.1: Configuración Señales Periódicas S0 – S7**